CANopen-controlled personnel interlock system at DESY

Rüdiger Härtel, port GmbH
Stefan May, DESY

A personnel interlock system is a safety critical part in an accelerator. It switches off all relevant radiation sources in case of danger, e.g. klystrons, magnets and particle sources. The personnel interlock system at DESY is a logical circuit made up of relays that are hardwired to emergency off switches, access doors, safety key boxes, beam shutters and other dry contacts. The interlock software monitors each relay with a current monitor (optocoupler) and a bottom contact. To operate an accelerator, a warning procedure with optical and acoustical warnings is required. The interlock software controls the warning procedure by influencing the current path in the relay circuit. If any of the signals going into the relay circuit is missing, beam operation is denied or withdrawn. All safety critical paths are realized with the relays only, the interlock software must not have any safety critical tasks. Besides the safety critical aspect, personnel interlock systems must provide high availability to ensure long operation times. This applies especially to the interlock software and its components.

CANopen based modules are used by the personnel interlock system to monitor the relays with a current monitor and a bottom contact. The personnel interlock software had to be overhauled from ground up. The new system had to be integrated into the existing interlock hardware infrastructure and makes use of standard CANopen means for configuration of CANopen modules (CiA-302-2) and process image processing (CiA-405). This paper discusses the used system architecture and aspects of integrating CANopen to Java.

Introduction

The German Electron Synchrotron DESY, Member of the Helmholtz Association, is one of the leading accelerator centers in the world in the research fields structure of matter and photon science. DESY is a national research center supported by public funds and has locations in Hamburg and Zeuthen (Brandenburg). DESY develops and builds large facilities for the acceleration of particles to extremely high energies.

A personnel interlock system is used to ensure safety for researchers and staff working at the accelerator. It permits and restricts access to the inner bounds of the accelerator ring. Before operation of the accelerator is permitted a special warning procedure is carried out to check if all signals report OK state.

The system described in this paper is used at the moment in the accelerators Linac2 and DESY2 and comprises

- 4 CAN lines,
- 38 CANopen I/O nodes,
- 254 relays and
- 2 field computers.

Other accelerators at DESY will be equipped with this system in future.

The radiation does not allow to place electronical device directly with in the accelerator. The radiation would cause defects in the silicon of the processors that are used in the devices. Therefore all signals are hardwired to relays outside of the accelerator. This is an import aspect for the safetyness of the system.

The existing system at DESY was to be overhauled and to be replaced with a system that makes better use of the communication mechanisms CANopen provides. The new system consists of

- a CANopen Manager,
- a Process Controller (CANopen slave) providing the process image to the control application through a Java binding and
- several I/O modules.

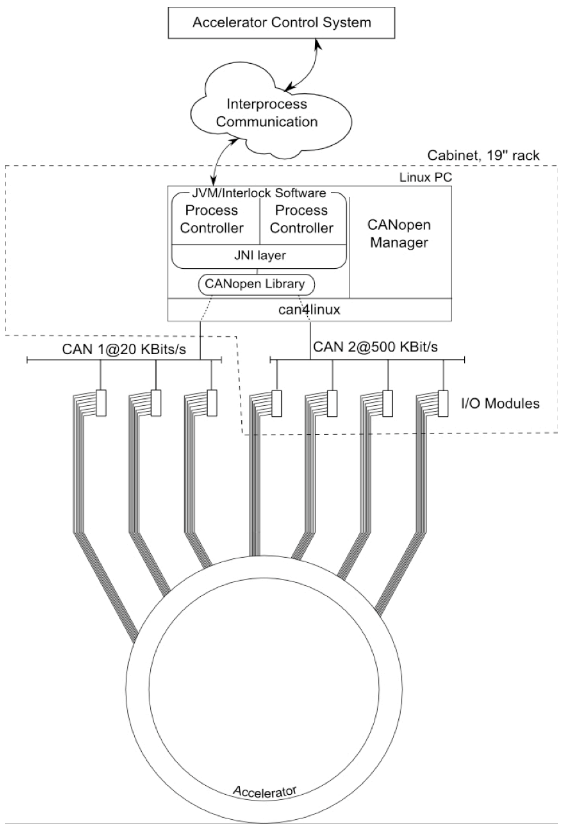


Figure 1: Application Overview

The software for the CANopen Manager and the Process Controller runs on an industrial PC with multiple CAN interfaces. The can4linux was extended to allow one CAN interface to be used by more than one process. The used operating system is Linux.

CAN Communication

The preferred communication principle of the CANopen protocol is asynchronous communication driven by events of the devices in an decentralized infrastructure. This approach is followed as closely as possible in the new system.

The Network Management (NMT-Master) is in most cases used within the control application. This combination is often used but not required. With the decentralized approach the NMT functionality was separated into an own programme which fulfills the task of device configuration and starting the network.

The functionality of collecting process data, emergency data and node monitoring is another programme which is a normal CANopen slave, the Process Controller.

I/O modules provide the digital and analogue signals (inputs) as process data (inputs) and use process data to write to digital and analogue outputs.

The majority of CANopen devices and the Linux PC are all placed in the same cabinet. The bus length is short and a bitrate of 500 KBit/s is used. Other CANopen devices like blinking lamps or tableaus are placed in the field. Due to the long distances (some hundred meters) the used baudrate is 20 KBit/s.

The safetyness of the signals is provided by the hardwiring. The CAN bus itself is not considered as safety critical. No special precautions for safe transmission like the CiA 304 are used. For data consistency PDO are configured as asynchronous with an event timer.

I/O Modules

CANopen has a large base of device profiles. The modules used in this system conform to the profile CiA 401. This guarantees exchangibility of the modules and independency from a single I/O module manufacturer.

Process Controller

The connection between the I/O modules and the interlock software is established by the Process Controller (figure 1). Process data, emergency messages, SDO accesses to the local object dictionary and node failures are passed through method calls over JNI to the interlock software. For the CANopen Manager the Process Controller is a normal slave like any of the I/O module. From application perspective can be seen as the master.

For the group MPS at DESY, Java is the standard programming language. The CANopen library which is implemented in C needed to be interfaced with Java [ShengLiang]. Instead of making a 1:1 mapping of the C API to a Java API the Java API was inspired by the function blocks from the CiA 405 to access the CANopen network.

designed in that way that all asynchronous events are reported via *update* methods. Emergency data is provided in its own class *EmergencyError*. A heartbeat error provides the node id and node state. For PDO mapped objects *update* is called separately for each object. For sending a new data item with a PDO the Java application writes a new value to the *CanOpenObjectDictionary*. The CANopen object is searched in the C implementation of the CANopen object dictionary and then sent.

Actively the Process Controller does not use SDO for communication with I/O modules. It only receives data via PDO from the I/O modules. Write accesses to the object dictionary are handled like PDO. For each object that is written to a update method is called. The Java application will not know if the update was caused by a PDO or a SDO.

`CIA405_STATE`	NodeState
`CIA405_EMCY_ERROR`	EmergencyError

Table 1: Mapping of CiA405 data types to Java

`CIA405_GET_STATE`	CanOpenObjectDictionary.getNode(id)
`CIA405_LOCAL_NODE`	CanOpenObjectDictionary.localNode()
`CIA405_RECV_EMCY`	CanOpenObjectDictionary.update(EmergencyError)

Table 2: Mapping of CiA405 function blocks to Java

The Java class *PortCanOpenObjectDict-ionary* provides access to all CANopen objects of the Process Controller itself that are available in the C implementation. CANopen objects are *OdEntry* and can be addressed by index, subindex or by Para-meterName and Denotation respect-ively. The information of the parameter names or denotation is read from the device configuration file (DCF). A new value for a *OdEntry* can be set with the method *update*.

The CiA 405 defines a function block only for the emergency event. Other asynchro-nous events like heartbeat error or PDO have no function block. The Java API was

Through the use of the *CanOpenObjectDict-ionary* as a process image an abstraction from the underlying technology is achieved. In principle it would be possible to replace the CANopen network by any other network technology.

CANopen Manager

The I/O modules and the Process Controller are simple CANopen slaves that do not have information about the complete network. The CANopen Man-ager has all information about the network. That means it knows which devices should

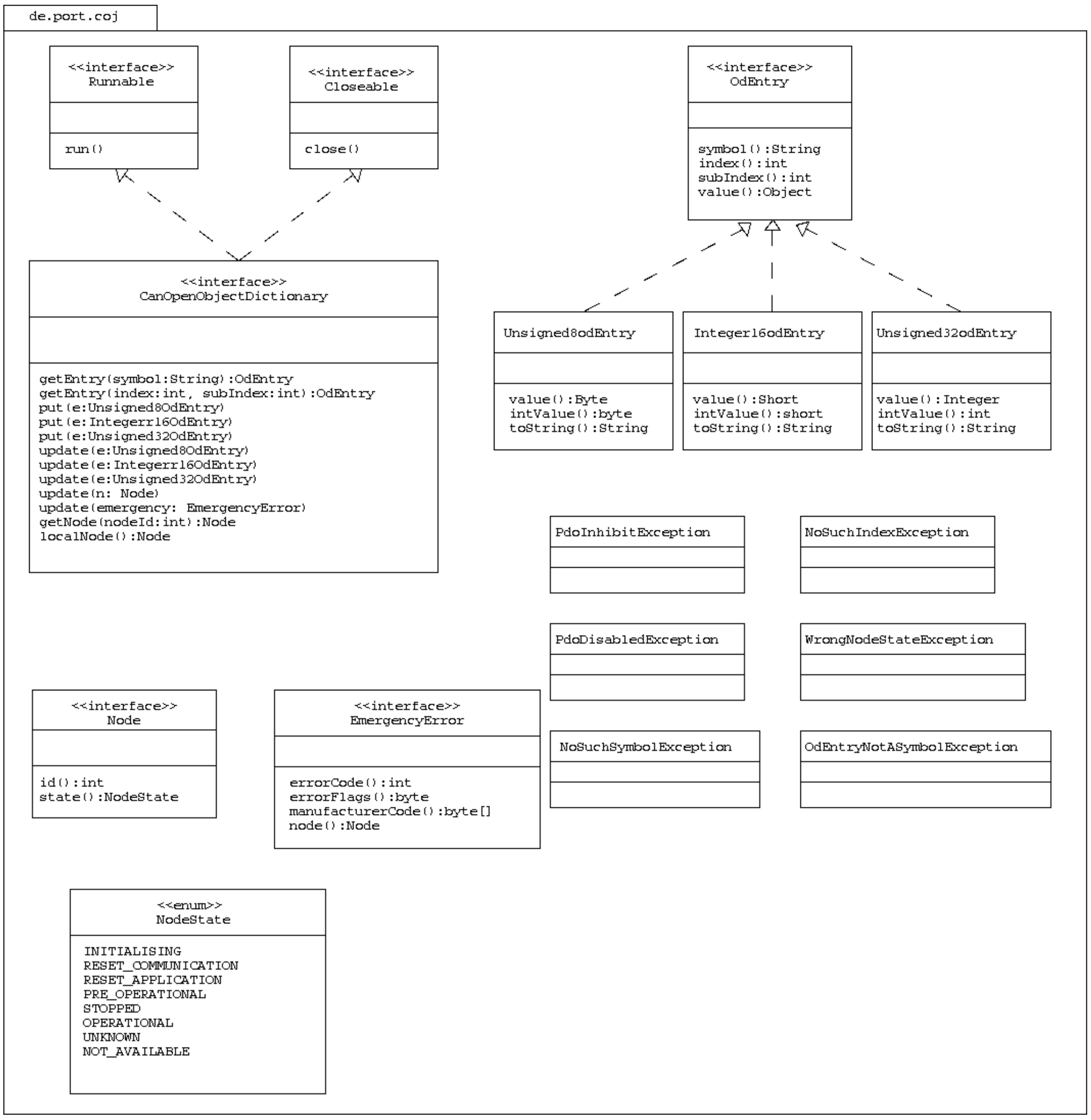


Figure 2: Type interface of the Process Controller

be in the network and which shouldn't. It also knows the configuration of each device.

As such it has different tasks to fulfill:

- log CANopen events
- orderly startup of devices
- distribute configuration
- check configuration regularly

The CANopen Manager uses 3 of the 8 logging levels of the syslog daemon of Linux. The level LOG_INFO is used for events that have informational character like configuration complete, heartbeat of node started and all slaves booted. Events that cause a logging entry with the level LOG_WARNING are heartbeat of node lost, an SDO access timed out or the like. LOG_ERR is used for emergency messages and unexpected internal events.

The orderly startup of devices is described thoroughly in the standard CiA 302-2. Starting a device comprises several steps:

- validating the device type
- validating the device identity
- validating device configuration

When it is verified that the configuration needs to be updated the CANopen dictionary of the devices is filled with the values from the device configuration files (DCF). The DCF of all devices is read at startup and stored in object 1F22 in concise format. This object is configured as readonly to prohibit changes from a configuration tool.

During the normal operation time the configuration of each node is read out via SDO and compared with the DCF value. This happens to report unintentional changes that may occur during a servicing like replacing a device. In case that a real value differs from the DCF value the device is not reconfigured only a log message is written via the syslog daemon.

Timing Considerations

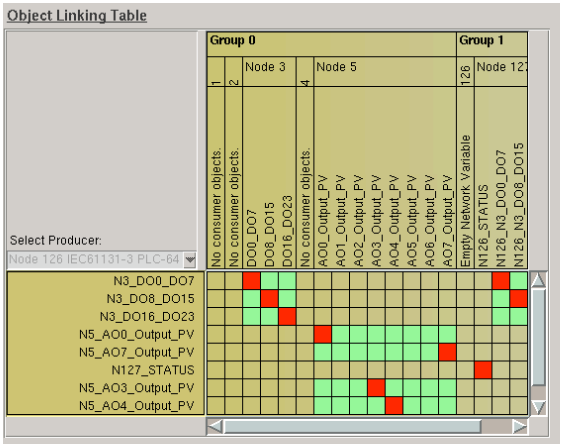
The application process is under normal conditions not very busy since nobody will enter the inner bounds of the accelerator when an experiment is run. Thus the bus load is generated only by the continuous SDO checks and the PDO timer events.

It is planned to provide I/O modules with object 1020 Verify Configuration. With the use of this object checking the configuration of a single device can be carried out with a single SDO access. This change allows to modify the continuous configuration check to a interval configuration check and thus lower the bus load further.

Configuration Management

As with the Process Controller and CANopen Manager the configuration management should be done with tools work conforming to the CANopen standard. CANopen devices come with an EDS that describes the parameters of the devices and thus the functionality of the device. The format is described in CiA 306. The configuration that had to be done was PDO linking, setting heartbeat and parameterization of inputs and outputs like inversion of the raw value. The result is a set of device configuration files which are used by the CANopen Manager.

It turned out that configuration tools can handle small to medium sized EDS files. The EDS file of the Process Controller had large an EDS file with over 3 MB. Configuration Tools from different manufacturers were tested but processing such a large file was not acceptable. Either it took a long loading time and the programme reacted slowly on user actions. This problem was not totally removed.


Object Linking Table

Select Producer:
Node 126 IEC61131-3 PLC-64

	Group 0														Group 1			
	1	2	Node 3			4	Node 5								126	Node 12		
	No consumer objects.	No consumer objects.	DO0_DO7	DO8_DO15	DO16_DO23	No consumer objects.	AO0_Output_PV	AO1_Output_PV	AO2_Output_PV	AO3_Output_PV	AO4_Output_PV	AO5_Output_PV	AO6_Output_PV	AO7_Output_PV	Empty Network Variable	N126_STATUS	N126_N3_DO0_DO7	N126_N3_DO8_DO15
N3_DO0_DO7			🟥	🟩	🟩												🟥	🟩
N3_DO8_DO15			🟩	🟥	🟩												🟩	🟥
N3_DO16_DO23			🟩	🟩	🟥												🟩	🟩
N5_AO0_Output_PV							🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩				
N5_AO7_Output_PV							🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥				
N127_STATUS																🟥		
N5_AO3_Output_PV							🟩	🟩	🟩	🟥	🟩	🟩	🟩	🟩				
N5_AO4_Output_PV							🟩	🟩	🟩	🟩	🟥	🟩	🟩	🟩				

Figure 3: Configuration with a matrix

Another problem when creating a configuration for such large networks is that tools cannot provide the overview that is needed. The data that should be exchanged between modules is either represented in a matrix (figure 3) or in a point to point connection. In both cases only a small cutout from the network is available. This is a source of errors. The problem can be tackled if the signals could be exported from the signal plan.

Conclusion and future development

The personnel interlock was inspected by the DESY radiation protection group D3 and approved to run for a year. Until now it is running stable.

Future improvements for the process controller are protection of the CANopen object dictionary against configuration changes when it is in Operational and executing a callback before changing to Operational. The CANopen Manager will be enhanced with a service mode to reset configured devices to the delivery state. The configuration of the system was very time consuming and thus is not optimal. The suggested improvement: generating a configuration based on a signal plan would release the staff from this error-prone work.

Rüdiger Härtel
port GmbH
Regensburger Straße 7b
06132 Halle
http://www.port.de
hae@port.de

Stefan May
DESY
Notkestrasse 85
22607 Hamburg
http://www.desy.de
stefan.may@desy.de

References

[1] CAN in Automation, Application Layer and Communication, Profile CiA 301 (04 January 2006).
[2] CAN in Automation, Additional application layer functions, Part 2 Network management CiA 302-6 (08 August 2006).
[3] CAN in Automation, Electronic Data Sheet Specification for CANopen CiA 306 (04 December 2002).
[4] CAN in Automation, Interface and Device Profile for IEC 61131-3 Programmable Devices CiA 405 (21 May 2002).
[5] port GmbH, CANopen, ANSI C library, User Manual (22 February 2006). ISBN 3-8334-4621-8.
[6] Addison-Wesley, Cay.S Horstmann and Gary Cornell, Core Java 2 (2005). ISBN 3-8273-22-16-2.
[7] Addison-Wesley, Sheng Liang, The Java Native Interface,
[8] Programmer's Guide and Specification (June 1999). ISBN 0-201-32577-2.